**n-Queens Problem Project**

**Yangqi Su 800957989**

The **n-Queens** problem is formulated as such:

Given a **n** x **n** chess board, place **n** Queens on the board such that no Queen can attack one another (a conflict is called each time 2 queens can attack one another). To solve the problem, the board is first initialized with a queen in each column randomly, thus one only needs to avoid conflicts between the rows and diagonals.

**Here we use a hill-climbing technique called steepest ascent which searches for the best move by calculating a heuristic for each queen to move within its own column.**

The python program submitted contains the **nQueenMatrixAgent** class:

The **nQueenMatrixAgent class** takes as initiation arguments a **heuristic** choice of ‘conflicts’ (currently only heuristic possible) or others (not implemented), along with whether the **size** of the problem: number of queens, and whether initiating the board should be done with **smart\_init** (essentially each column and row has only one queen in the first step).

It then can searches for a solution with the **HillClimbingSolver** function which takes arguments such as whether to use **sideway** moves and whether **random restarts** are used and **number of restarts** if restarts are allowed. If a solution is found or if the program failed, the program outputs a tuple of the following format:

**(binary indicator indicating success or failure,**

**history of the search process,**

**number of iterations used (equals 1 for non-restart methods),**

**number of moves made ( for each iteration) )**

The **conflicts** function calculates total conflicts for the entire board or number of conflicts for a particular position:

The number of conflicts for the entire board is calculated by calculating the **sigma sum** for each row sum, left diagonal sum and right diagonal sum of the board configuration matrix. the matrix contains only 1s and 0s, 1s representing the Queen. For instance, if the sum of a row is 6, then **the sigma sum of that row is: 5+4+3+2+1=15**, which is the number of conflicts in that row between each pair of queens. The goal state is reached if the total number of conflicts for a board configuration is 0.

The number of conflict for moving a queen to a position within its own column, is calculated as the sum of the current row, left diagonal and right diagonal corresponding to that position.

The **updownMoves** function calculates for each queen the heuristic cost of moving within their respective columns and choose the queen with the best heuristic as the queen to move. If sideway moves are enabled, then the move can be to a position that costs the same as the current position if no better move is found.

**Global Variables** are 100 rounds for each test, and n=8 for examples of the 8-queen problem. The n can be supplied in the manner described below as well.

**The results are provided within a Jupyter Notebook for convenient viewing. The python script provided can be used as follows:**

**#** num\_queens is the number of queens provided by the user **#**

**python nQueens.py num\_queens**